BO·TT·EGA
IT minds

Program szkolenia:

Mikroserwisy, Python i DDD - kompendium

Informacje:

Nazwa:	**Mikroserwisy, Python i DDD - kompendium**
Kod:	**python-ddd5**
Kategoria:	Python
Odbiorcy:	architekci, developerzy
Czas trwania:	5 dni
Forma:	40% wykłady, 60% warsztaty

Kompleksowe, 5-dniowe szkolenie z budowania rozproszonych systemów informatycznych w Pythonie w oparciu o Domain-driven design. Szkolenie kładzie nacisk na zaprojektowanie modularności, a następnie integrowanie komponentów w niezawodny sposób.

Koncentrujemy się na rozwiązywaniu rozmaitych problemów związanych z komunikacją asynchroniczną oraz poznawanie wzorców wspomagających - jak Saga czy CQRS.

Ostatniego dnia zajmujemy się tematyką monitorowania z klasycznymi metrykami oraz Distributed Tracing. Uzupełnimy też wiadomości z testowania, zapewniając jakość w budowanym systemie.

- Dzień 1: Wyznaczanie granic i projektowanie modularności
- Dzień 2: Wzorce taktyczne DDD
- Dzień 3: System rozproszone i architektura sterowana zdarzeniami
- Dzień 4: Niezawodność w systemach rozproszonych
- Dzień 5: Testowanie, monitorowanie i zapewnienie jakości

Zalety szkolenia:

- Pragmatyczne podejście
- Szukanie granic autonomicznych modułów
- Sprawdzone wzorce architektoniczne

BOTTEGA IT minds

Szczegółowy program:

1. Wprowadzenie do Domain-Driven Design

- 1.1. Dlaczego potrzebujemy modularności?
  - 1.1.1. Utrzymywalność oprogramowania
  - 1.1.2. Skalowanie wytwarzania oprogramowania
  - 1.1.3. Łatwiejsze rozbijanie projektu na mniejsze
  - 1.1.4. Pogodzenie różnych perspektyw
  - 1.1.5. Pogodzenie różnych interesariuszy
- 1.2. Jak DDD pomaga modularyzować?
  - 1.2.1. Jeden model to z mało…
  - 1.2.2. …zamiast tego używamy kilku, wokół których organizujemy usługi / komponenty
  - 1.2.3. DDD, a organizacja zespołów

2. Event Storming - analizujemy system

- 2.1. Zastosowania Event Stormingu
- 2.2. Poziomy Event Stormingu
- 2.3. Myślenie zdarzeniami
- 2.4. Big Picture - pierwszy rzut oka na system z lotu ptaka
  - 2.4.1. Zdarzenie
- 2.5. Process Level Event Storming - odkrywamy procesy i proponujemy Bounded Contexty
  - 2.5.1. Komenda
  - 2.5.2. Aktor
  - 2.5.3. Hotspot

3. DDD i modularność w praktyce - szukamy Bounded Contextów

- 3.1. Poddomeny - problem space

BOTTEGA IT minds

3.2. Bounded Contexty - solution space

3.3. Heurystyki znajdowania Bounded Contextów

3.4. Budujemy mapę kontekstów

4. Drivery architektoniczne

4.1. Specyfika technologii

4.2. Specyfika biznesowa

4.3. Wymagania funkcjonalne

4.4. Wymagania niefunkcjonalne

4.5. Inne

5. Inne heurystyki wspomagające projektowanie modularności

5.1. Zgodność z regulacjami

5.2. Security

5.3. Częstość zmian

5.4. Ryzyko

5.5. Wydajność

5.6. Model współbieżności

6. Zapachy architektury świadczące o nieoptymalnym designie

6.1. Cyclic dependency

6.2. Feature / data envy

6.3. God component

6.4. Dense Structure

7. Projektujemy system składający się z mikroserwisów

7.1. Założenia

7.1.1. Projektowanie z myślą o awariach

7.1.2. Podejście ewolucyjne

BOTTEGA IT minds

7.1.3. Decentralizacja zarządzania danymi

7.2. Jak określić granicę serwisów

7.2.1. Podejście Bounded Context z DDD

7.2.2. Antywzorzec: Nanoservice

7.2.3. Antywzorzec: Rozproszony CRUD

8. Pułapki czyhające przy projektowaniu systemu rozproszonego

8.1. Współdzielona baza danych

8.2. Nadużywanie synchronicznej komunikacji

8.3. Pojedynczy punkt awarii

8.4. Źle wyznaczone granice

8.5. Brak możliwości niezależnego wdrożenia

9. Agregaty

9.1. Heurystyki projektowania

9.1.1. Gwarancja spójności reguł biznesowych

9.1.2. Granica spójności transakcji

9.1.3. Preferowanie małych agregatów

9.1.4. Przechowywanie tożsamości innych agregatów

9.1.5. Używanie eventual consistency do synchronizacji stanu agregatów

9.2. Kiedy łamiemy zasady? Na co uważać?

9.3. Walidowanie zaprojektowanych agregatów

9.3.1. Upewnienie się, że problemy z początku szkolenia nam nie grożą

10. Value objects

10.1. Odpowiednie do modelowania konceptów

11. Serwisy Domenowe

11.1. Model procedur biznesowych

BO·TT·EGA
IT minds

12. Fabryki

- 12.1. Walidacja
- 12.2. Logika biznesowa podczas składania obiektów
- 12.3. Wsparcie testowalności

13. Polityki (strategie)

- 13.1. Modelowanie w stylu funkcyjnym
- 13.2. Open Close Principle (SOLID) w praktyce

14. Zdarzenia dziedzinowe

- 14.1. Wyzwalanie akcji w ramach tego samego Bounded Contextu
- 14.2. Event Bus
- 14.3. Integrowanie Bounded Contextów

15. Czysta architektura / Porty i Adaptery

- 15.1. Logika orkiestracji - Przypadek użycia / Serwis aplikacyjny
- 15.2. Logika zmiany - encje oraz wzorce taktyczne z DDD
- 15.3. Logika walidacji - specyfikacja, DTO

16. Integrowanie Bounded Contextów

- 16.1. Kierunek zależności
  - 16.1.1. Komendy
  - 16.1.2. Zdarzenia
- 16.2. Anticorruption Layer
- 16.3. Open Host / Published Language
- 16.4. Customer - Supplier
- 16.5. Shared Kernel

17. Komunikacja synchroniczna - API

- 17.1. REST API

17.1.1. Podstawy projektowania API

17.1.2. Model dojrzałości Richardsona

17.1.3. Dokumentowanie na przykładzie OpenAPI

17.2. Wady i pułapki komunikacji synchronicznej

17.2.1. Zwiększanie podatności na awarie

17.2.2. Temporal coupling

18. Komunikacja asynchroniczna - Wiadomości

18.1. Asynchroniczne komendy

18.2. Zdarzenia

19. CQRS

19.1. Ogólne założenia

19.1.1. Wertykalny podział

19.1.2. Model zapisu

19.1.3. Model odczytu

19.2. Przykładowe zastosowania

19.2.1. widok dla panelu administracyjnego

19.2.2. raportowanie

19.2.3. audit log

19.3. Read model tylko na poziomie kodu

19.4. Budowanie read modeli - synchronicznie

19.5. Budowanie read modeli - asynchronicznie

19.6. Radzenie sobie z opóźnieniem odświeżenia read modelu

20. Wzorce Sagi i Process Managera

20.1. Orkiestracja kontra choreografia w architekturach sterowanych zdarzeniami

20.2. Stanowy Process Manager czy bezstanowa Saga?

BOTTEGA IT minds

20.3. Zarządzanie długim procesem

20.4. Kompensacja

20.5. Obsługa timeoutów

21. Gwarancje dostarczenia

21.1. Najczęściej uzyskiwana - at-most-once

21.2. Najczęściej potrzebna - at-least-once

21.3. Najczęściej pożądana - exactly-once

21.4. Wybór właściwej gwarancji

22. Outbox Pattern

22.1. Pułapka wysłania za wcześnie - przed zatwierdzeniem transakcji

22.2. Pułapka wysłania później - po zatwierdzeniu transakcji

22.3. Gwarancja wysłania wiadomości

22.4. Implementacja z relacyjną bazą danych

22.5. Skalowanie z relacyjną bazą danych

23. Radzenie sobie z nieprawidłowymi wiadomościami

23.1. Walidacja

23.2. Odrzucanie

23.3. Kolejka na nieprawidłowe wiadomości

24. Dead Letter i finalizacja wiadomości

24.1. realizacja na wybranym brokerze

24.2. ponowne próby

24.2.1. backoff

24.2.2. ilość maksymalna

24.3. polityka finalizacji i retencji nieskonsumowanych wiadomości

BOTTEGA IT minds

25. Wzorzec Idempotent Receiver i deduplikacja wiadomości

- 25.1. Ignorowanie duplikatów
- 25.2. Ponowne przetworzenie tych samych wiadmości
- 25.3. Strategie deduplikacji
  - 25.3.1. Identyfikator wiadomości
  - 25.3.2. Wersja
  - 25.3.3. Idempotentna wiadomość

26. Ewolucja i wersjonowanie zdarzeń

- 26.1. Potrzeba wersjonowania
- 26.2. Serializacja wspierające ewolucję - Apache Avro, Protobuf, JSON Schema
- 26.3. Przykład implementacji Schema Registry
- 26.4. Jak wdrażać serwisy, by bezpiecznie migrować do nowszych wersji?

27. Testowanie

- 27.1. Testowanie akceptacyjne serwisu / komponentu
- 27.2. Testy end-to-end
- 27.3. Testy kontraktowe
  - 27.3.1. Consumer-driven Contract Testing
- 27.4. Rodzaje test doubles i ich wpływ na test
  - 27.4.1. Mock
  - 27.4.2. Stub
  - 27.4.3. Dummy
  - 27.4.4. Fake
- 27.5. Wzorce wspomagające
  - 27.5.1. Fabryki
  - 27.5.2. Warstwy w testach, DSL

BOTTEGA IT minds

27.6. Testy syntetyczne

28. Metryki

28.1. Prometheus i Grafana

28.2. Wsparcie w popularnych bibliotekach

28.3. Budowanie własnych metryk

28.4. Co warto mierzyć? Co dostajemy za darmo?

29. Distributed Tracing

29.1. opentelemetry

29.1.1. Biblioteki

29.1.2. Wsparcie

29.1.3. Własna instrumentacja

29.1.3.1. Dodawanie własnych danych do trace'ów

29.2. Wizualizacja przepływów w systemie

29.3. Znajdowanie źródła problemu wydajnościowego